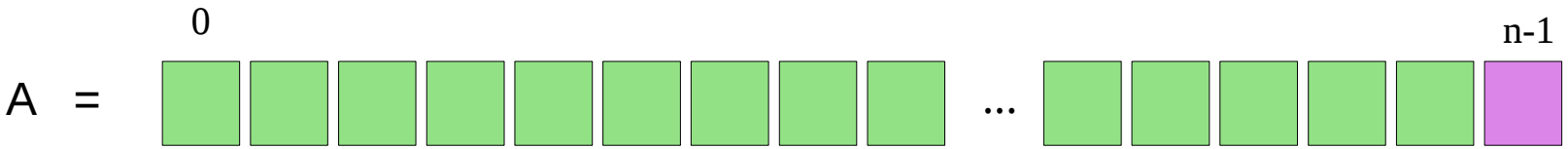
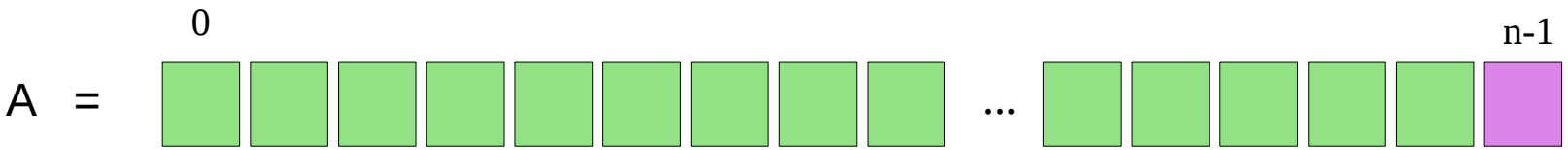
Aula 19
Custos de um algoritmo e funções de complexidade

Prof. Jesús P. Mena-Chalco

3Q-2017



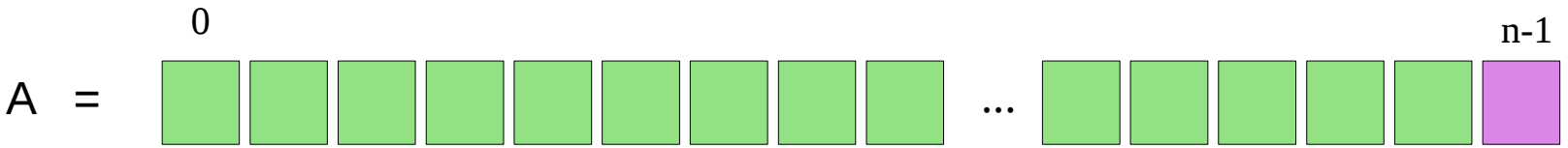

```c
int buscaChave(int chave , int A[], int n) {
    int i;

    for(i=0; i<n; i++) {
        if (chave == A[i])
            return i;
    }
    return -1;
}
```




```c
int buscaChave(int chave , int A[], int n) {
    int i;

    for(i=0; i<n; i++) {
        if (chave == A[i])
            return i;
    }
    return -1;
}
```

- O programa funciona (está correto)?

- Como medir/mensurar a eficiência
(em termos de tempo e espaço) do programa?



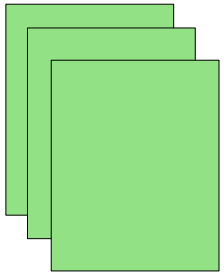

```c
int buscaChave(int chave , int A[], int n) {
    int i;

    for(i=0; i<n; i++) {
        if (chave == A[i])
            return i;
    }
    return -1;
}
```

- O programa funciona (está correto)?

- Como medir/mensurar a eficiência (em termos de tempo e espaço) do programa?

Análise de algoritmos

AED1
Análise de algoritmos



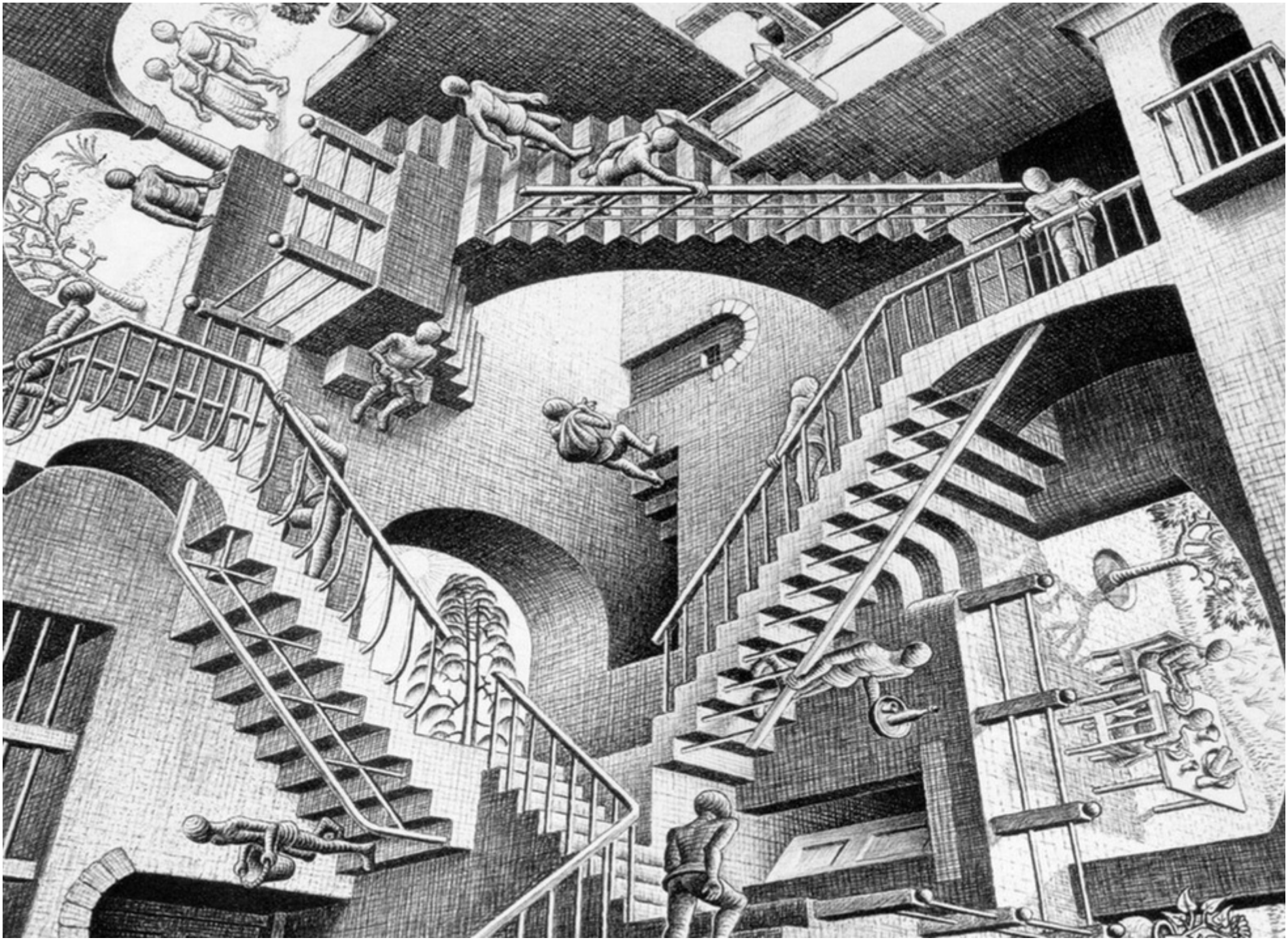
1997

2017

Estudo de algoritmos

- O projeto de algoritmos **é influenciado pelo estudo de seus comportamentos**.

- Os algoritmos podem ser **estudados** considerando, entre outros, dois aspectos:
  - Tempo de execução.
  - Espaço ocupado (quantidade de memória).



Fonte: http://hqwallbase.com/images/big/stairways-1514922.jpg

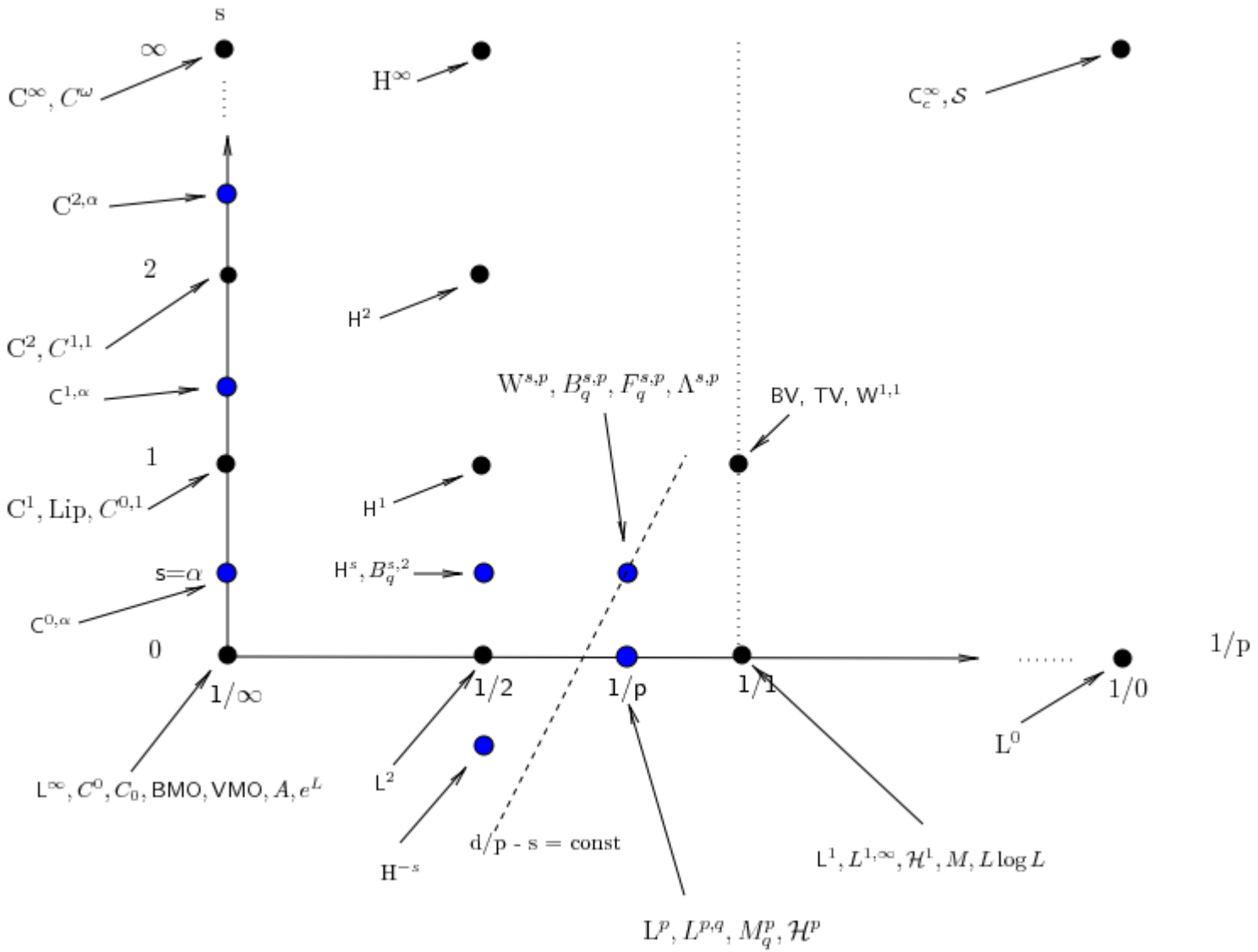
Medida de custo pela execução de um programa em uma plataforma real

- Tais medidas são bastante inadequadas e os resultados jamais devem ser **generalizados**:
  - Os resultados são **dependentes do compilador** que pode favorecer algumas construções em detrimento de outras;
  - Os resultados **dependem de hardware**;
  - Quanto grandes quantidades de memória são utilizadas, as medidas de tempo podem depender deste aspecto.

Medida de custo pela execução de um programa em uma plataforma real

- Tais medidas são bastante inadequadas e os resultados jamais devem ser generalizados:
  - Os resultados são **dependentes do compilador** que pode favorecer algumas construções em detrimento de outras;
  - Os resultados **dependem de hardware**;
  - Quanto grandes quantidades de memória são utilizadas, as medidas de tempo podem depender deste aspecto.

- Apesar disso, há argumentos a favor de se obterem medidas reais de tempo:
  - Exemplo: Quando há vários algoritmos distintos para resolver o problema;
  - Assim, são considerados tanto os custos reais das operações como os custos não aparentes, tais como alocação de memória, indexação, carga, dentre outros.


s
∞
C^∞, C^ω
H^∞
$C_c^\infty, \mathcal{S}$
$C^{2,\alpha}$
2
$C^2, C^{1,1}$
H^2
$W^{s,p}, B_q^{s,p}, F_q^{s,p}, \Lambda^{s,p}$
BV, TV, $W^{1,1}$
$C^{1,\alpha}$
1
C^1, Lip, $C^{0,1}$
H^1
$H^s, B_q^{s,2}$
s=α
$C^{0,\alpha}$
0
1/∞
1/2
1/p
1/1
1/0
1/p
L^0
L^∞, C^0, C_0, BMO, VMO, A, e^L
L^2
H^{-s}
d/p - s = const
$L^1, L^{1,\infty}, \mathcal{H}^1, M, L \log L$
$L^p, L^{p,q}, M_q^p, \mathcal{H}^p$

Comparando algoritmos?

```
int F1(int a, int b) {
    int i, t1, t2;

    t1 = a;
    t2 = b;

    a = t2;
    b = t1;
```

```
int F2(int a, int b) {
    int i, t;

    t = a;

    a = b;
    b = t;
```

Comparando algoritmos?

```
int F1(int a, int b) {
    int i, t1, t2;

    t1 = a;
    t2 = b;

    a = t2;
    b = t1;
```

```
int F2(int a, int b) {
    int i, t;

    t = a;

    a = b;
    b = t;
```

	F1	F2
Número de linhas	5	4
Número de variáveis	5	4
Número de atribuições	4	3
Número de comparações entre elementos	0	0

Exercício 1

Exercício 1

```c
#include "stdio.h"

int main(void) {

  int i, n, soma1, soma2;

  n     = 100;
  soma1 = 0;
  soma2 = 0;

  for (i = n; i >= 0; i = i-1) {
    soma1 = soma1 + i;
    soma2 = soma2 - i;
  }

  printf("%d  %d", soma1, soma2);

  return 0;
}
```

```
5050  -5050
```

Exercício 1

```c
#include "stdio.h"

int main(void) {

  int i, n, soma1, soma2;

  n     = 100;
  soma1 = 0;
  soma2 = 0;

  for (i = n; i >= 0; i = i-1) {
    soma1 = soma1 + i;
    soma2 = soma2 - i;
  }

  printf("%d  %d", soma1, soma2);

  return 0;
}
```

	programa
Número de linhas	
Número de variáveis	
Número de atribuições	
Número de comparações entre elementos	

Exercício 1

```c
#include "stdio.h"

int main(void) {

  int i, n, soma1, soma2;

  n     = 100;
  soma1 = 0;
  soma2 = 0;

  for (i = n; i >= 0; i = i-1) {
    soma1 = soma1 + i;
    soma2 = soma2 - i;
  }

  printf("%d  %d", soma1, soma2);

  return 0;
}
```

	programa
Número de linhas	13?
Número de variáveis	4
Número de atribuições	3n+4
Número de comparações entre elementos	n+2

Exercício 2

Exercício 2

```
int buscaChave(int chave , int A[], int n) {
    int i;

    for(i=0; i<n; i++) {
        if (chave == A[i])
            return i;
    }
    return -1;
}
```

	buscaChave
Número de linhas	
Número de variáveis	
Número de atribuições	
Número de comparações entre elementos	

Exercício 2

```c
int buscaChave(int chave , int A[], int n) {
    int i;

    for(i=0; i<n; i++) {
        if (chave == A[i])
            return i;
    }
    return -1;
}
```

	buscaChave
Número de linhas	6? 8?
Número de variáveis	4? 3+n?
Número de atribuições	n+1
Número de comparações entre elementos	n? 2(n)+1?

Depende do que?

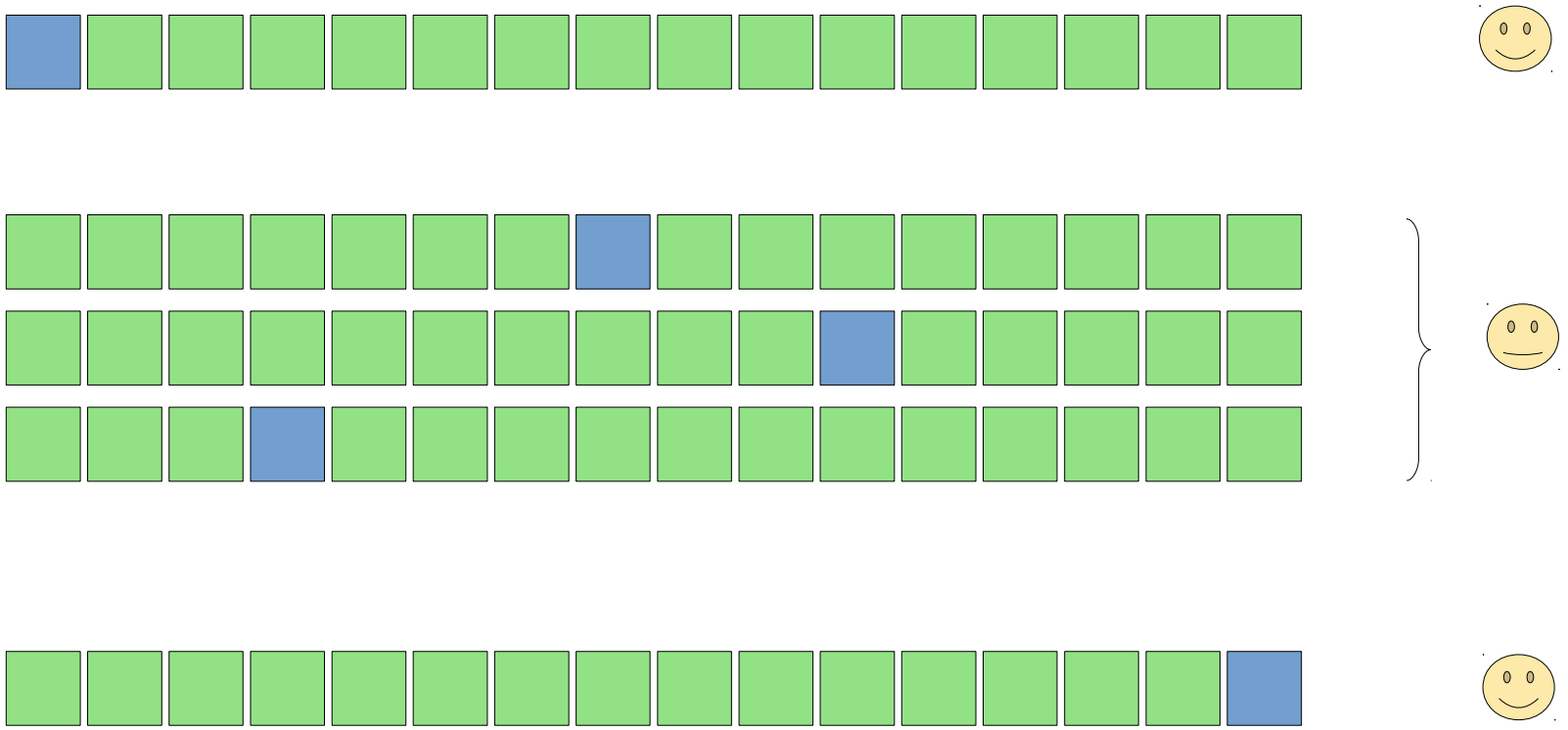
Exercício 2

```c
int buscaChave2(int chave , int A[], int n) {
    int i;
    i = n-1;

    while(i>=0 && A[i]!=chave) {
        i-=1;
    }
    return i;
}
```

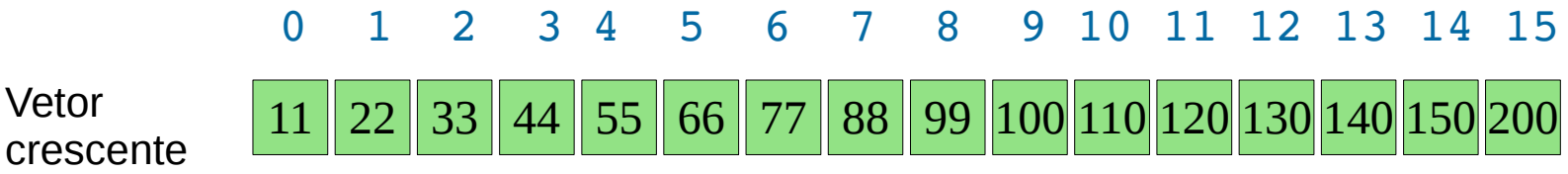
```c
int buscaChaveRec(int chave , int v[], int n) {
    if (n==0)
        return -1;
    if (chave == v[n-1])
        return n-1;
    return buscaChaveRec(chave, v, n-1);
}
```

	buscaChave	buscaChave2	buscaChaveRec
Número de linhas	8?	8?	7?
Número de variáveis	4? 3+n?	4? 3+n?	3? 2+n?
Número de atribuições	(n+1)?	n+1?	0 ?
Número de comparações entre elementos	n? 2(n)+1?	2(n)?	**?**

no máximo?
no pior caso?



Busca de um elemento em um vetor crescente

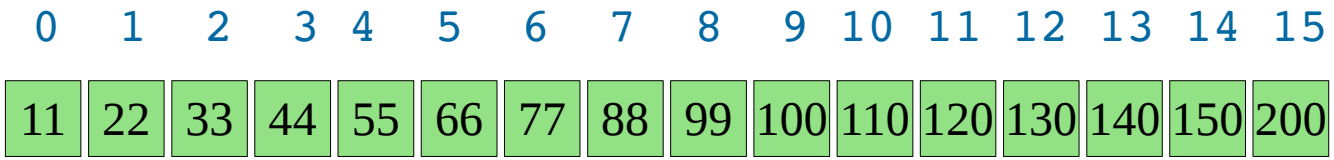


Custo para buscar um elemento em um vetor crescente:

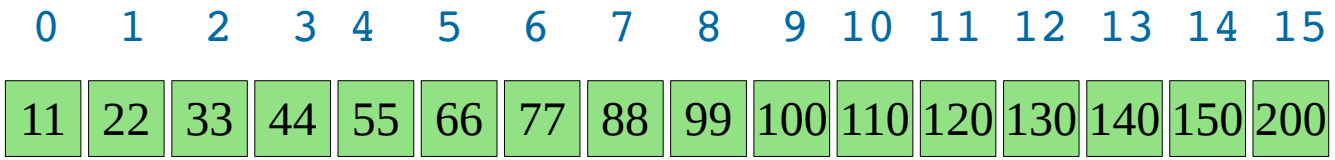
Melhor caso: 1

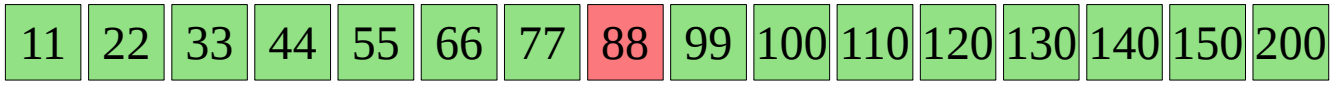
Pior caso: log(n) ?

Chave = 101


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
11 22 33 44 55 66 77 88 99 100 110 120 130 140 150 200

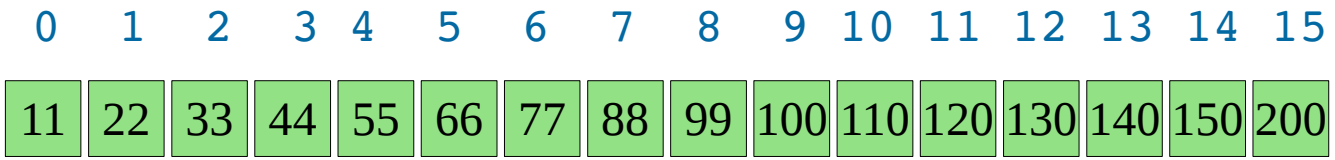
Chave = 101

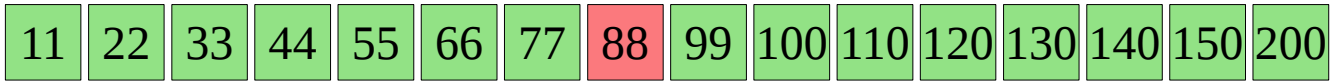




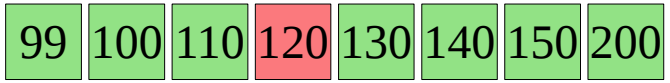
Inf = 0
Sup = 15

Chave = 101



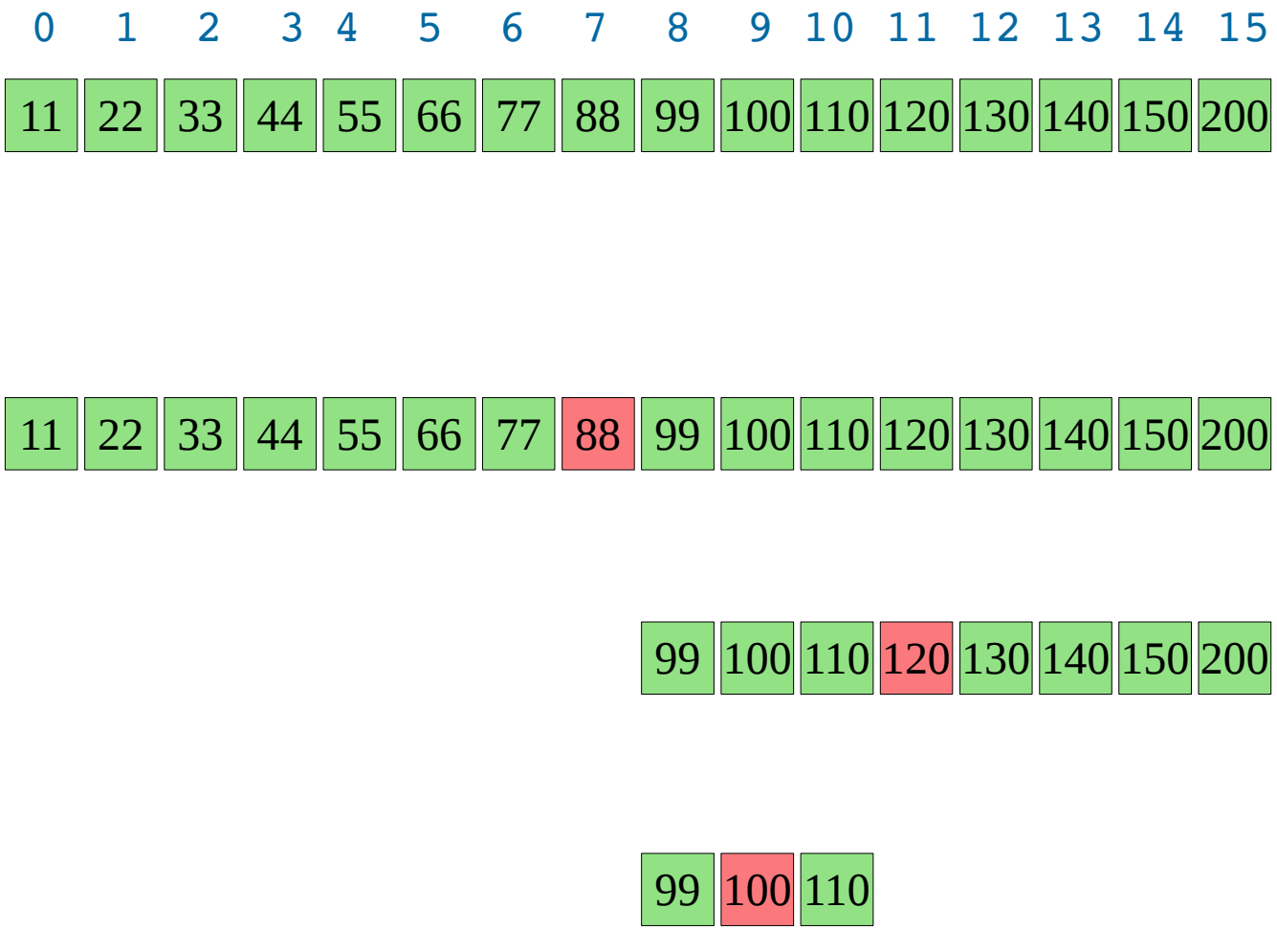


Inf = 0
Sup = 15

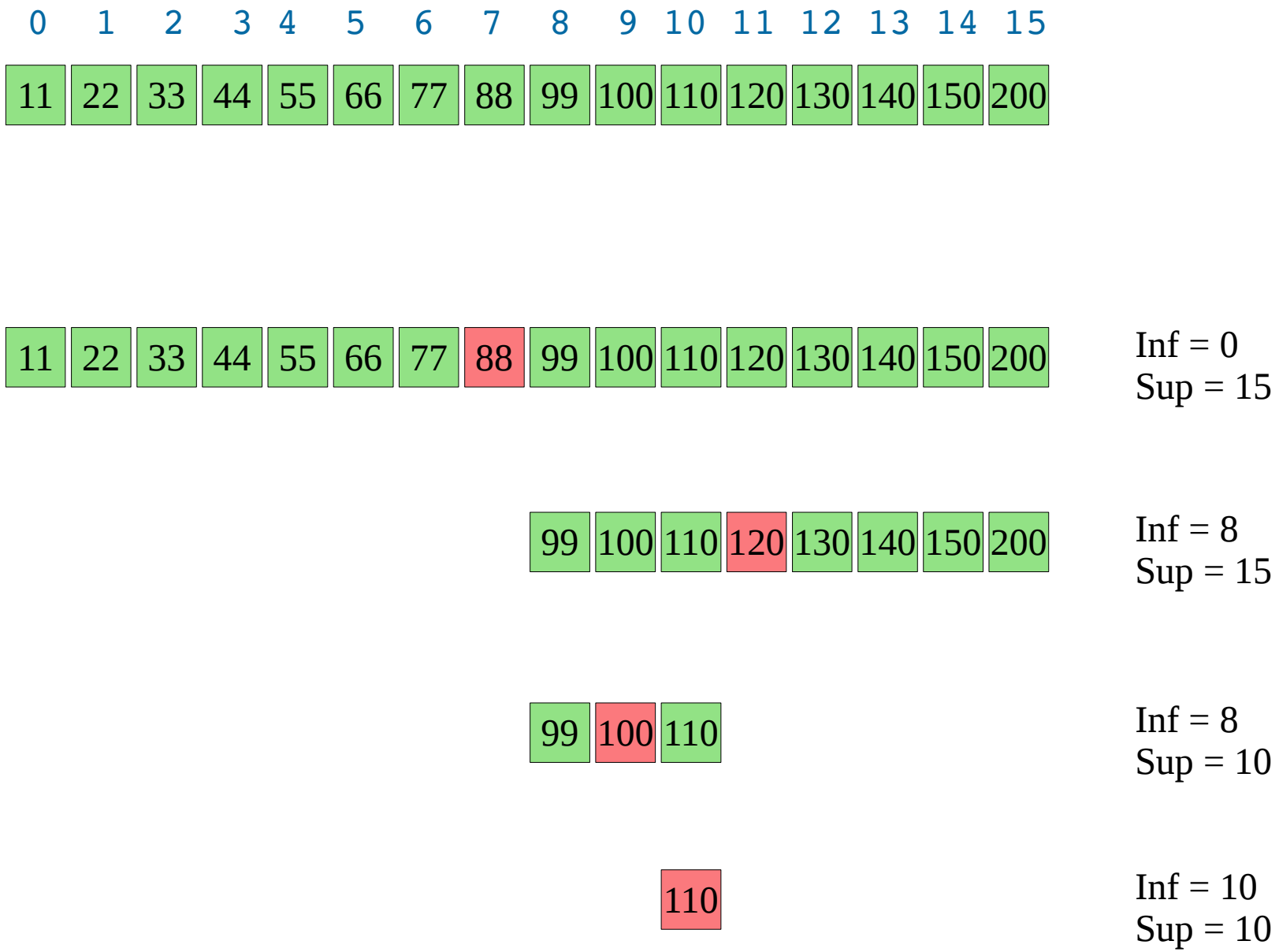


Inf = 8
Sup = 15

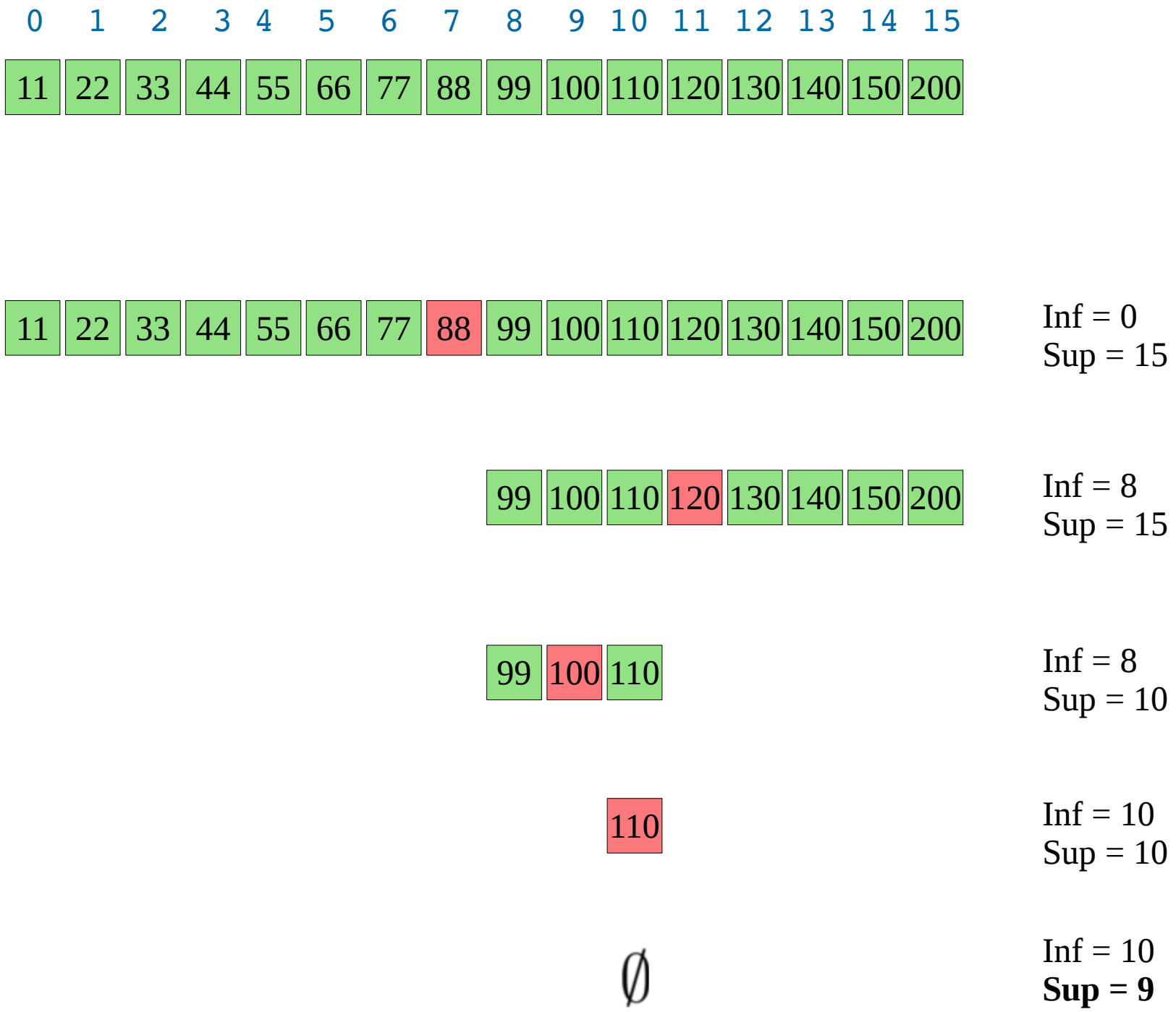
Chave = 101


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
11 22 33 44 55 66 77 88 99 100 110 120 130 140 150 200
11 22 33 44 55 66 77 88 99 100 110 120 130 140 150 200
Inf = 0
Sup = 15
99 100 110 120 130 140 150 200
Inf = 8
Sup = 15
99 100 110
Inf = 8
Sup = 10

Chave = 101


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
11 22 33 44 55 66 77 88 99 100 110 120 130 140 150 200
11 22 33 44 55 66 77 88 99 100 110 120 130 140 150 200
Inf = 0
Sup = 15
99 100 110 120 130 140 150 200
Inf = 8
Sup = 15
99 100 110
Inf = 8
Sup = 10
110
Inf = 10
Sup = 10

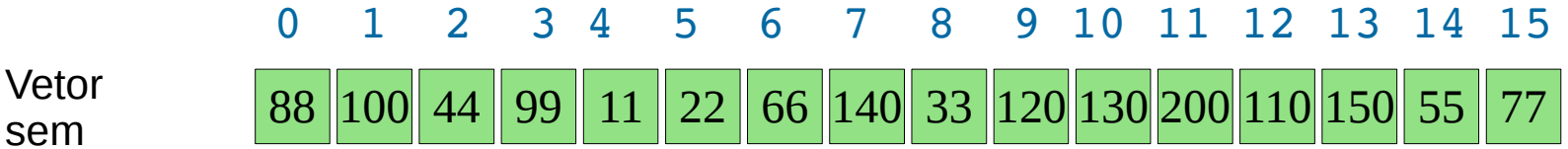
Chave = 101


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
11 22 33 44 55 66 77 88 99 100 110 120 130 140 150 200
11 22 33 44 55 66 77 88 99 100 110 120 130 140 150 200
Inf = 0
Sup = 15
99 100 110 120 130 140 150 200
Inf = 8
Sup = 15
99 100 110
Inf = 8
Sup = 10
110
Inf = 10
Sup = 10
∅
Inf = 10
Sup = 9

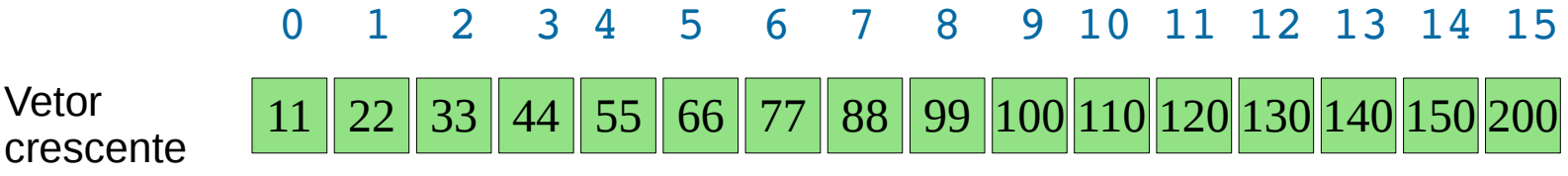
Busca de um elemento em um vetor crescente

```
int buscaBinariaRec (int chave, int A[], int inf, int sup) {
    int meio = (inf+sup)/2;

    if (inf>sup)
        return -1;

    if (chave==A[meio])
        return meio;

    if (chave<A[meio])
        return buscaBinariaRec(chave, A, inf, meio-1);
    else
        return buscaBinariaRec(chave, A, meio+1, sup);
}
```

	buscaBinariaRec
Número de linhas	?
Número de variáveis	?
Número de atribuições	?
Número de comparações entre elementos	3 lg(n)?



Melhor caso: 1
Pior caso: n



Melhor caso: 1
Pior caso: log(n)

N vs LG(N)

n	lg(n)
2	1
32	5
512	9
8192	13
131072	17
2097152	21
33554432	25
536870912	29
8589934592	33
137438953472	37
2199023255552	41
35184372088832	45
562949953421312	49
9007199254740990	53
144115188075856000	57
2305843009213690000	61
36893488147419100000	65
5.9029581035871E+020	69
9.4447329657393E+021	73
1.5111572745183E+023	77
2.4178516392293E+024	81
3.8685626227668E+025	85
6.1897001964269E+026	89
9.9035203142831E+027	93
1.5845632502853E+029	97
2.5353012004565E+030	101
4.0564819207303E+031	105
6.4903710731685E+032	109
1.0384593717070E+034	113
8.3076749736557E+034	116

Medida de custo por meio de um modelo matemático

- Usa um **modelo matemático** baseado em um **computador idealizado**.
- Deve ser especificado o **conjunto de operações e seus custos de execuções**.
- É mais usual ignorar o custo de algumas das operações e **considerar apenas as mais significantes.**
  - Em algoritmos de ordenação:

    Consideramos o conjunto de comparações entre os elementos do conjunto a ser ordenado e ignoramos as operações aritméticas, de atribuição e manipulação de índices, caso existam.

Função de complexidade

- Para medir o custo de execução de um algoritmo, é comum definir uma função de custo **ou função de complexidade *f*.**

- Função de **complexidade de tempo**:

  $f(n)$ mede o tempo necessário para executar um algoritmo para um problema de tamanho ***n***.

- Função de **complexidade de espaço**:

  $f(n)$ mede a memória necessária para executar um algoritmo para um problema de tamanho ***n***.

Utilizaremos ***f*** para denotar uma função de complexidade de tempo daqui para frente.

Na realidade, ***f*** não representa tempo diretamente, mas **o número de vezes que determinada operação (considerada relevante) é realizada**.

Atividade em aula

ATIVIDADE 01: Hierarquias de funções

$$\frac{N^2}{2} - \frac{N}{2}$$

$$\frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$$

$$\lg N + 1$$

$$\frac{N(N+1)}{2}$$

ATIVIDADE 01: Hierarquias de funções

Ordem de crescimento

$\frac{N^2}{2} - \frac{N}{2}$

$\frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$

$\lg N + 1$

$\frac{N(N+1)}{2}$

$\frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$

$\frac{N(N+1)}{2}$

$\frac{N^2}{2} - \frac{N}{2}$

$\lg N + 1$

Cúbico

Quadrático

Quadrático

Logaritmico

Maior hierarquia

Menor hierarquia

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$		*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$		*divide and conquer*	*mergesort*

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`
			double loop	*check all pairs*

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`
			triple loop	*check all triples*

exponential	2^N		*exhasutive search*	*check all subsets*

(*) Fonte: http://algs4.cs.princeton.edu/14analysis/

ATIVIDADE 02: Ordem de crescimento

```
int G1 (int N) {
    int n, i, sum=0;
    for (n=N; n>0; n/=2)
        for (i=0; i<n; i++)
            sum++;
    return sum;
}
```

```
int G2 (int N) {
    int i, j, sum=0;
    for (i=1; i<=N; i*=2)
        for(j=0; j<i; j++)
            sum++;
    return sum;
}
```

```
int G3 (int N) {
    int i, j, sum=0;
    for (i=1; i<=N; i*=2)
        for (j=0; j<N; j++)
            sum++;
    return sum;
}
```

ATIVIDADE 02: Ordem de crescimento

```
int G1 (int N) {
    int n, i, sum=0;
    for (n=N; n>0; n/=2)
        for (i=0; i<n; i++)
            sum++;
    return sum;
}
```

Linear

G1(N) = 2N-1

```
int G2 (int N) {
    int i, j, sum=0;
    for (i=1; i<=N; i*=2)
        for(j=0; j<i; j++)
            sum++;
    return sum;
}
```

Linear

G2(N) = 2N-1

```
int G3 (int N) {
    int i, j, sum=0;
    for (i=1; i<=N; i*=2)
        for (j=0; j<N; j++)
            sum++;
    return sum;
}
```

Linearithmic

G3(N) = N(lg(N)+1)

Bônus:
Limite assintótico para a ordenação

Ordenação

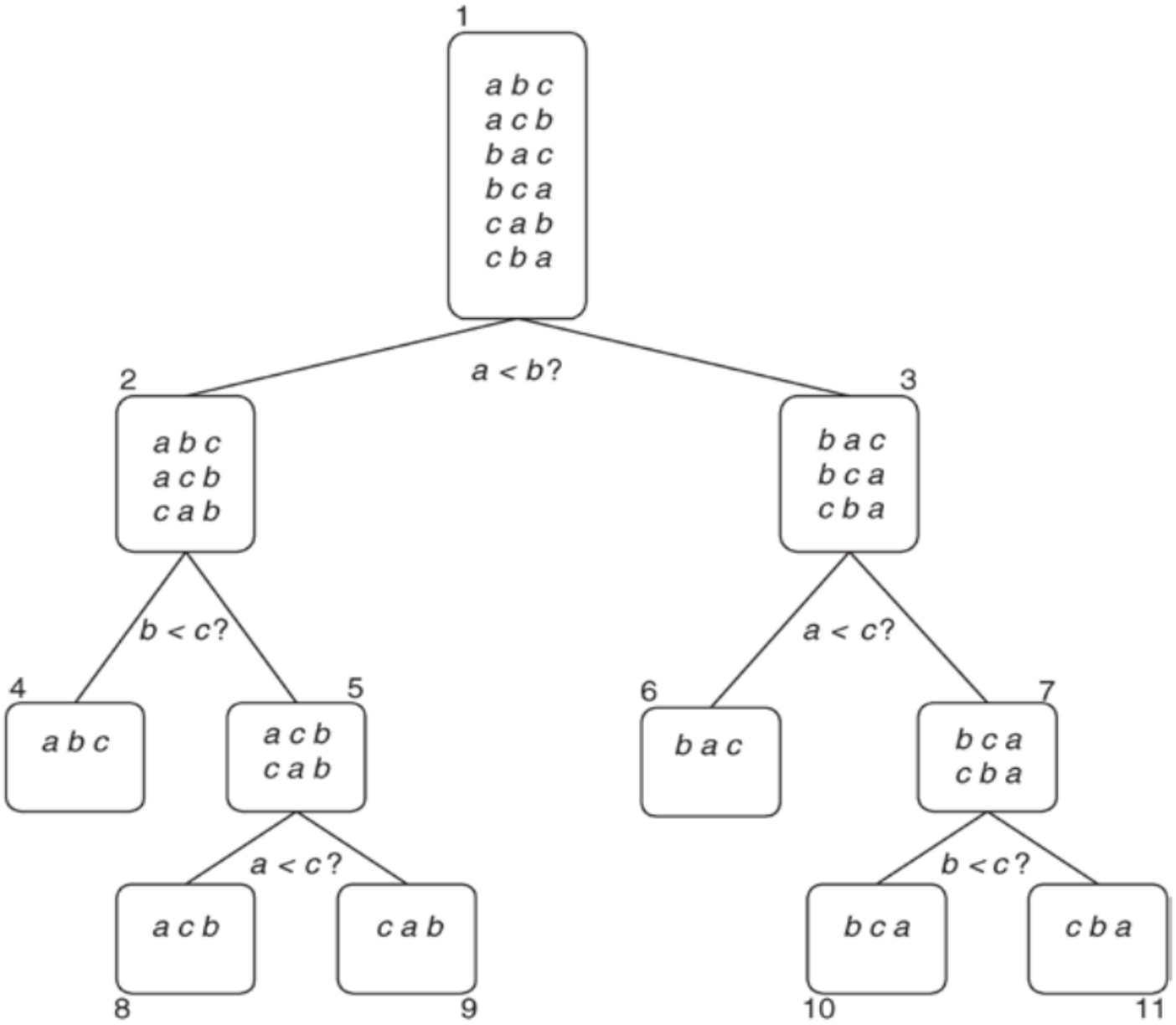
- **Algoritmos basedos em Comparações**
  - Insertion sort
  - Selection sort
  - Bubble sort
  - Merge sort
  - Quick sort
  - Quick Insertion sort

Complexidade computacional $\Omega(n \log(n))$

[limite matemático]
[limite assintótico para a ordenação]

[Árvore de decisão]

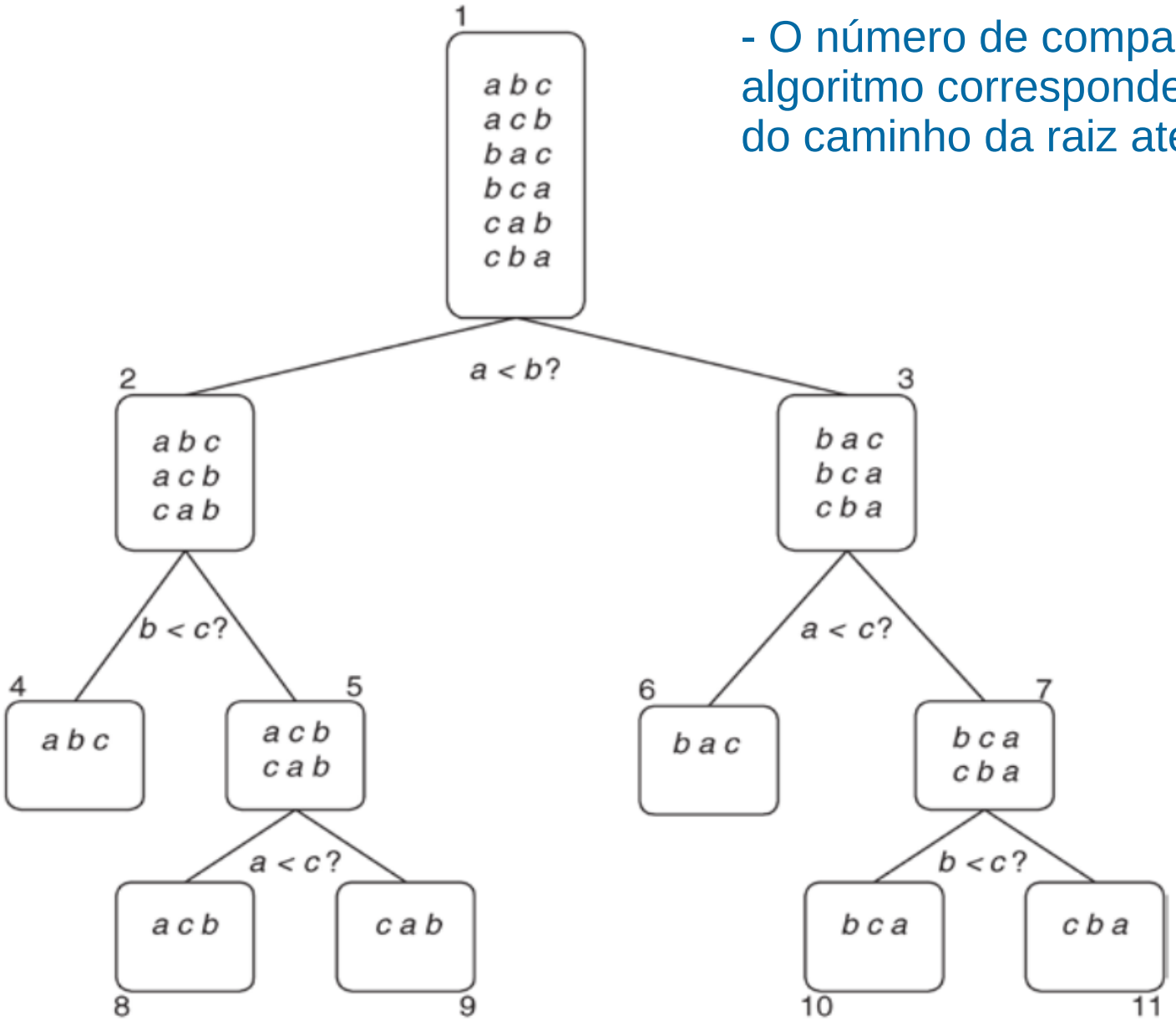
- Qualquer algoritmo de ordenação por comparação pode ser representado por uma árvore de decisão.

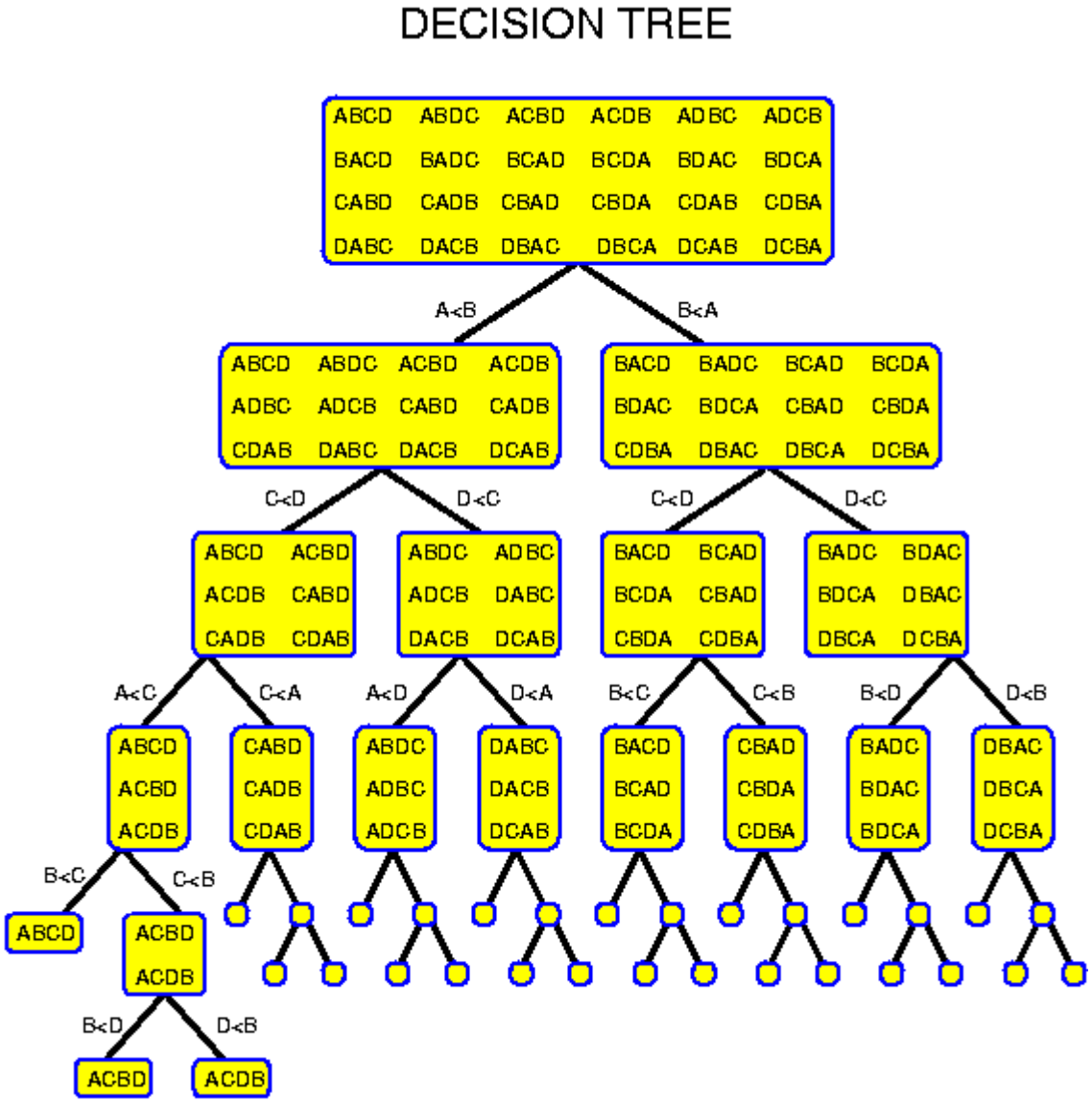


[Árvore de decisão]

- Qualquer algoritmo de ordenação por comparação pode ser representado por uma árvore de decisão.

- O número de comparações efetuadas pelo algoritmo corresponde ao **maior comprimento** do caminho da raiz até uma de suas folhas.



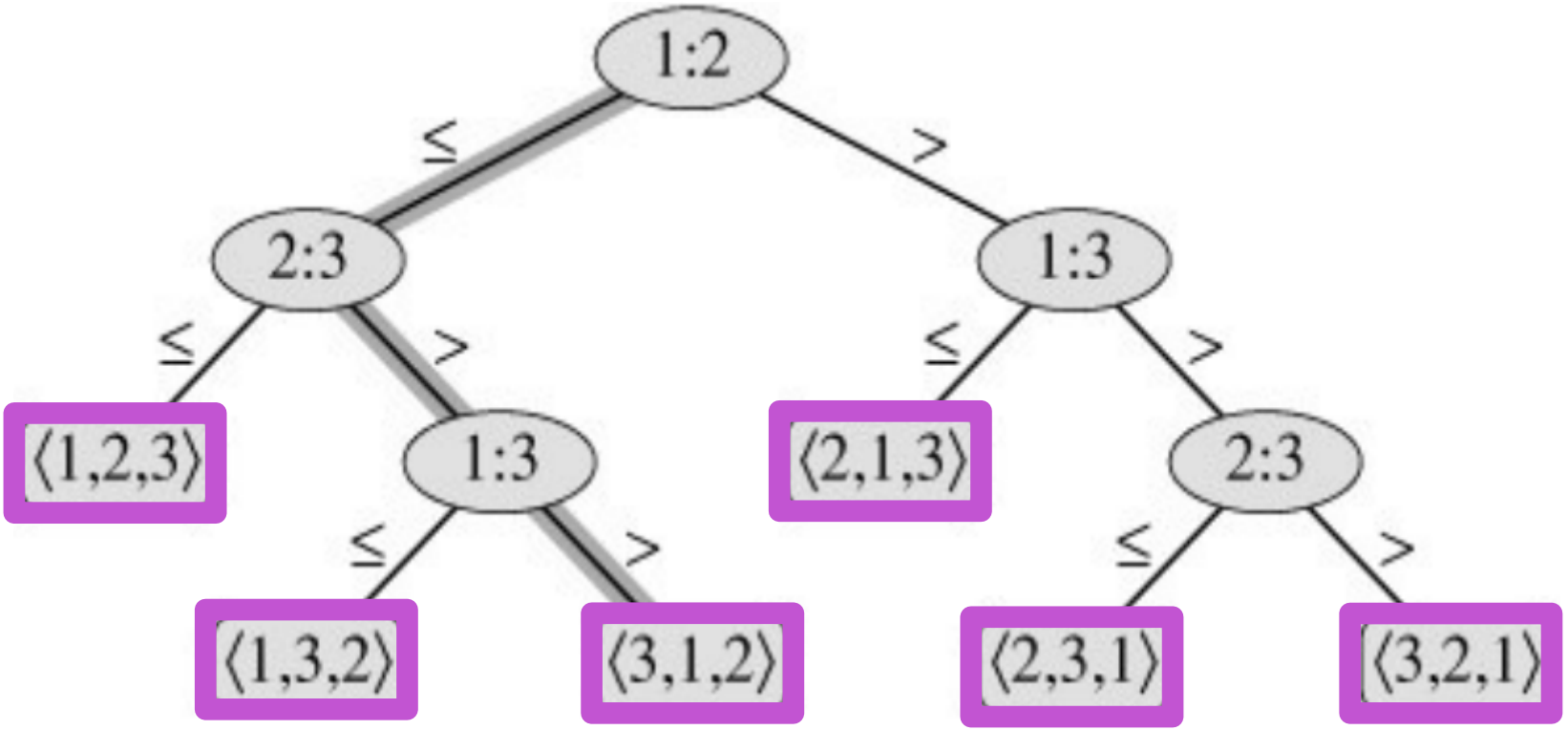


Fonte: http://users.informatik.uni-halle.de/~jopsi/dinf205/chap8.shtml

Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]

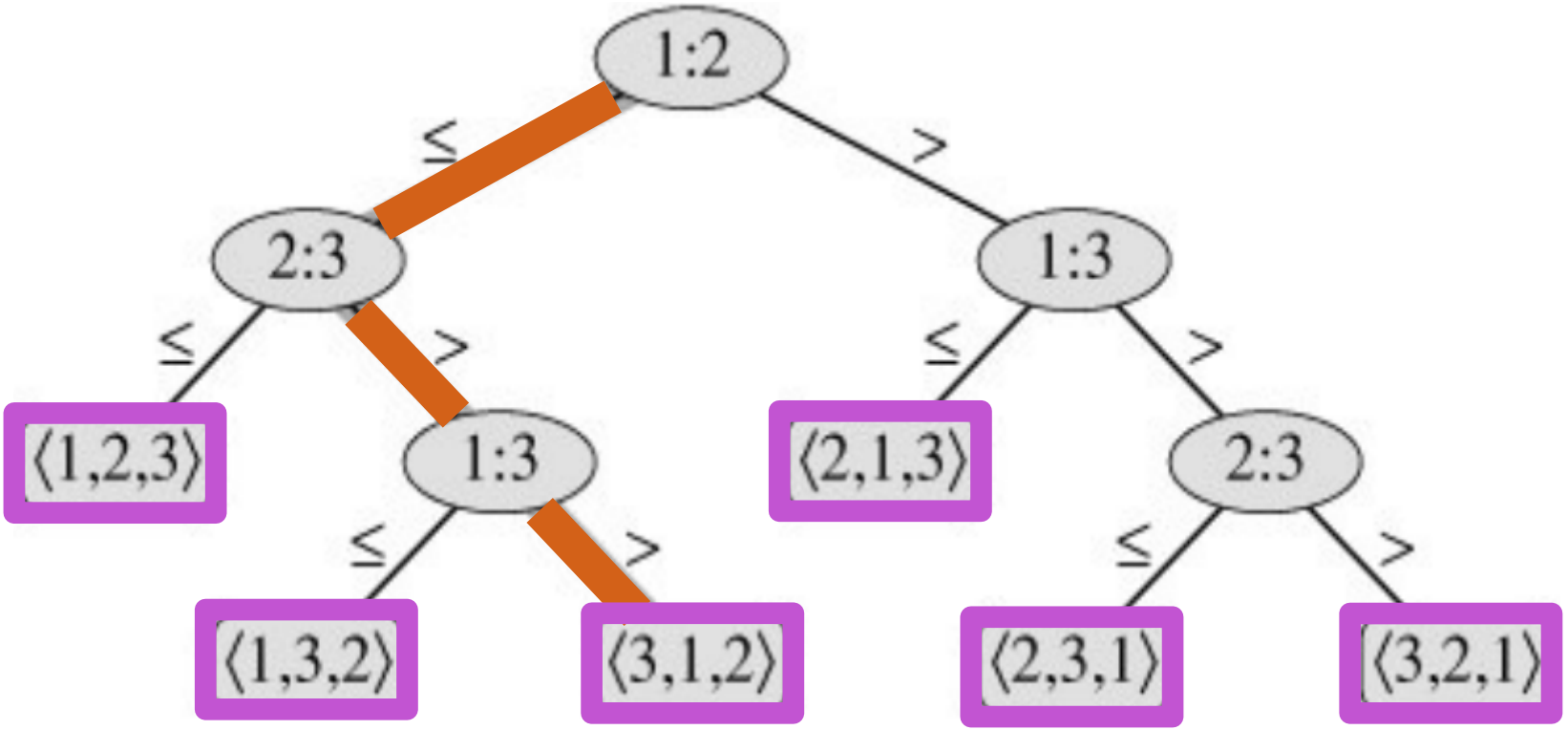


[Cada nó folha está associada a uma permutação dos elementos do vetor]

Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]



[Qualquer algoritmo de ordenação deverá percorrer um caminho desta árvore]

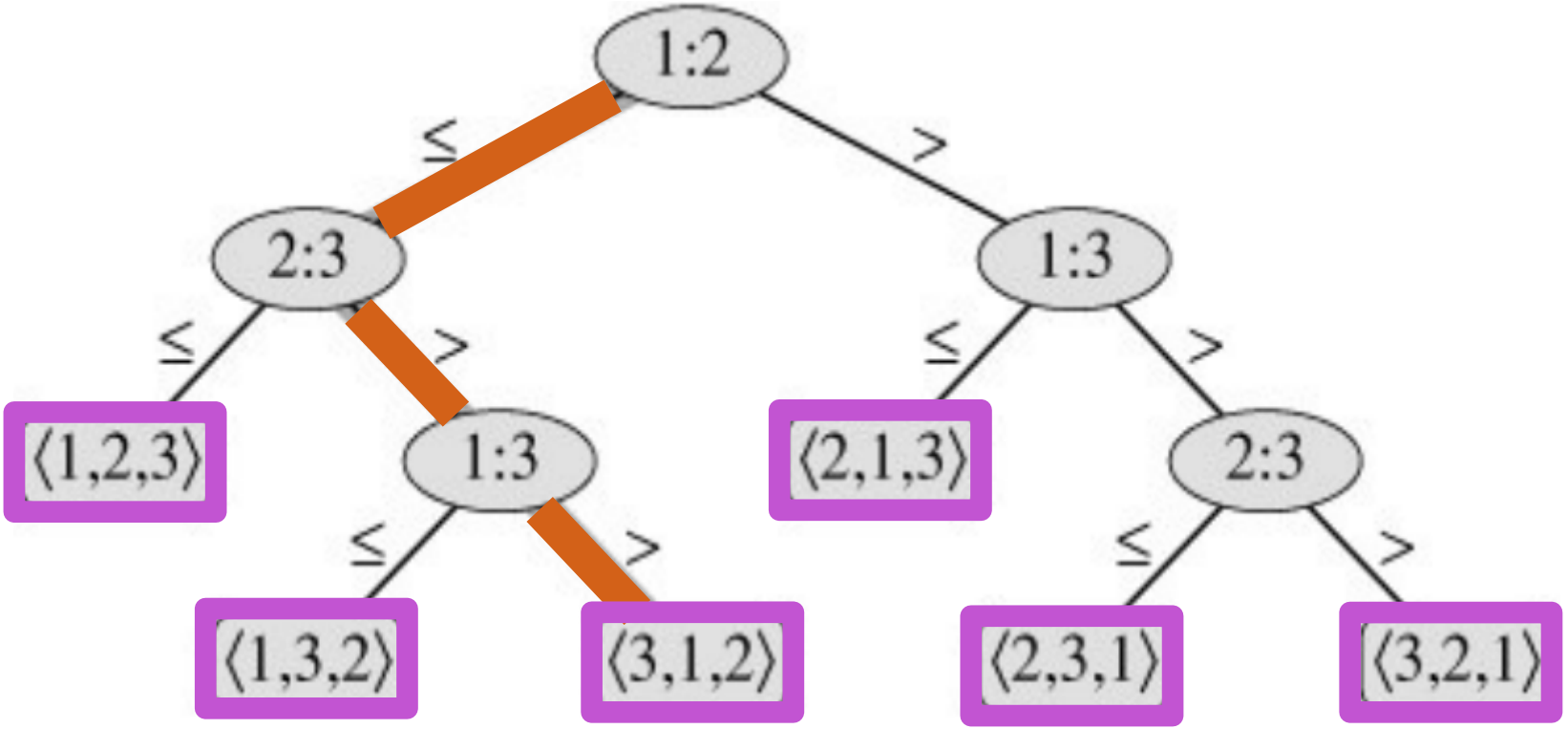
da raiz até a folha

Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]

Número de folhas = n!



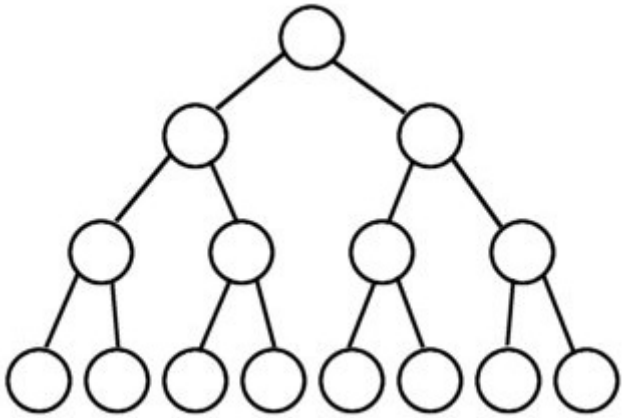
[Qualquer algoritmo de ordenação deverá percorrer um caminho desta árvore]

da raiz até a folha

Ordenação baseada em comparações

Seja ***L*** o número de folhas de uma árvore binária e ***h*** sua altura.

Então $L \leq 2^h$



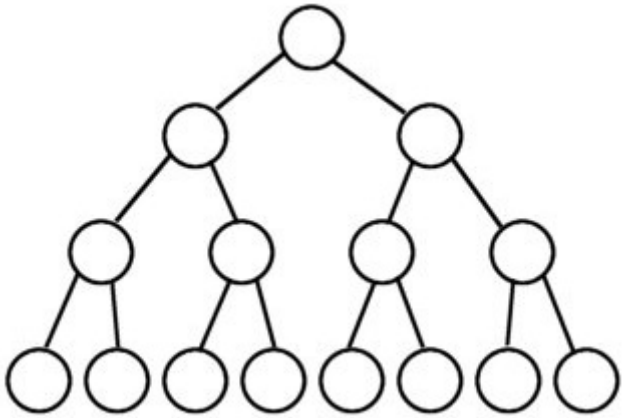
h=3

L=8

Ordenação baseada em comparações

Seja ***L*** o número de folhas de uma árvore binária e ***h*** sua altura.

Então $L \leq 2^h$



h=3

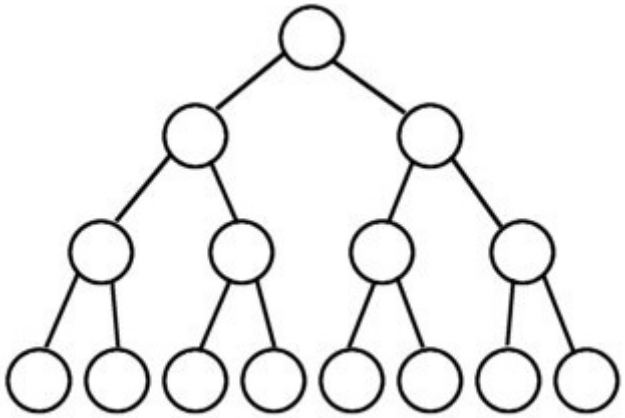
L=8

$h \geq \log(n!)$

Ordenação baseada em comparações

Seja ***L*** o número de folhas de uma árvore binária e ***h*** sua altura.

Então $L \leq 2^h$



h=3

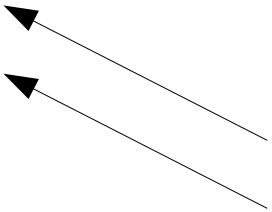
L=8

$$h \geq \log(n!)$$

$$h = \Omega(n \log n)$$

Ordenação baseada em comparações

- **Algoritmos basedo em Comparaçães**
  - Insertion sort
  - Selection sort
  - Bubble sort
  - Merge sort
  - Quick sort



Vários algoritmos aqui listados são ótimos pois a sua complexidade computacional é $O(n \log n)$